УДК 005.334

ІНСТРУМЕНТ АВТОМАТИЗОВАНОГО АНАЛІЗУ PRODUCTION COMPLIANCE NODE.JS ЗАСТОСУНКІВ

Драненко О.А.
e-mail: oleksii.dranenko@nure.ua
Науковий керівник – канд. техн. наук, доц. Любченко В.А.
Харківський національний університет радіоелектроніки, каф. ІНФ
м. Харків, Україна

This work presents a web-based tool for automated analysis of Node.js application production compliance. The system performs static analysis of GitHub repositories across three dimensions: security, reliability and performance, and code health. Each dimension is evaluated independently and contributes to an overall production compliance score. The tool targets developers who build applications using AI-assisted coding tools and lack experience in deploying production-grade software. Unlike existing solutions oriented toward enterprise environments, the proposed system provides actionable, automatically generated code fixes tailored to the Node.js ecosystem.

Великі мовні моделі (LLM) спричинили появу нового явища у сфері розробки програмного забезпечення – створення повноцінних застосунків особами без технічної освіти, які повністю делегують написання коду AI-інструментам таким як Claude, ChatGPT та GitHub Copilot. Серед платформ, що використовуються для таких застосунків, особливе місце займає Node.js – за даними звіту GitHub Octoverse 2025 [1], TypeScript вперше став найпопулярнішою мовою на GitHub, обігнавши Python та JavaScript, а більшість сучасних фреймворків генерують TypeScript кодову базу за замовчуванням, що робить Node.js природним вибором при AI-генерації.

Застосунки, створені таким чином, як правило, містять системні проблеми, що не проявляються під час розробки, але стають критичними після публічного розгортання. За даними галузевого звіту CodeRabbit [2], AI-згенерований код містить у 1.7 рази більше проблем порівняно з людським кодом, при цьому кількість проблем безпеки вища у 2.74 рази, а прогалини в обробці помилок – майже вдвічі.

Існуючі інструменти статичного аналізу коду, такі як Snyk, SonarQube та ESLint, орієнтовані на досвідчених розробників у корпоративному середовищі та потребують складного налаштування у складі CI/CD-пайплайну. При цьому навіть у професійному середовищі їх ефективність залишається обмеженою – перше емпіричне дослідження інструментів статичного аналізу для Node.js коду [3] протестувало дев’ять інструментів на датасеті з 957 вразливостей і встановило, що три найкращі разом виявляють лише 57.6% вразливостей при точності 0.11%. Таким чином, особа без технічної освіти залишається без жодного доступного та ефективного інструменту перевірки свого застосунку. У зв’язку з цим виникає потреба у

спеціалізованому інструменті, орієнтованому на простоту використання та автоматизацію аналізу.

Розроблена система являє собою веб-застосунок з GitHub OAuth-інтеграцією. На першому етапі виконується детекційне сканування: система визначає фреймворки, бази даних, ORM, засоби валідації, логування та тестування, а також структуру проєкту з точками входу для frontend і backend частин. Зібраний контекст дозволяє проводити глибокий Node.js-специфічний аналіз з урахуванням особливостей конкретних бібліотек – наприклад, Prisma, Mongoose або Express. Це дозволяє уникнути узагальнених перевірок і натомість застосовувати правила аналізу, адаптовані до конкретного технологічного стеку проєкту.

На другому етапі код аналізується за трьома вимірами production compliance:

–Безпека (Security) – валідація вхідних даних, rate limiting, налаштування CORS, захист від ін’єкцій, зберігання секретів;

–Надійність та продуктивність (Reliability & Performance) – пагінація, кешування, обробка помилок, health check ендпоінти, надмірні запити до БД;

–Якість коду (Code Health) – .gitignore, .env.example, логування, тести, зберігання стану.

Вибір саме цих трьох вимірів обґрунтований їх практичною значущістю. Вимір безпеки є критичним, оскільки вразливості в цій категорії безпосередньо загрожують даним користувачів та можуть призвести до несанкціонованого доступу до системи. Вимір надійності та продуктивності визначає здатність застосунку витримувати реальне навантаження – проблеми в цій категорії безпосередньо впливають на доступність сервісу та операційні витрати. Вимір якості коду, хоча і не створює прямих ризиків для кінцевого користувача в моменті, є предвісником майбутніх інцидентів – наприклад, відсутність коректного .gitignore призводить до витоку секретів, відсутність логування унеможливлює діагностику збоїв у робочому середовищі, а зберігання стану в пам’яті процесу гарантує втрату даних при перезапуску сервера.

Виявлення проблем реалізовано як дворівневий конвеєр, що поєднує швидкість детермінованого аналізу з точністю AI-верифікації. На першому рівні програмний аналізатор виконує синтаксичний та структурний аналіз кодової бази: він сканує AST-дерево, конфігураційні файли та патерни імпортів для виявлення підозрілих ділянок коду – відсутніх middleware, незахищених ендпоінтів, потенційних витоків секретів або неоптимальних запитів до бази даних. Результатом першого рівня є набір конкретних фрагментів коду з контекстом, що потребують подальшої перевірки. На другому рівні лише ці цільові фрагменти передаються до AI-моделі для підтвердження або спростування кожного потенційного порушення. Така архітектура забезпечує два ключові переваги: по-перше, значно вищу точність завдяки

зосередженню AI на конкретних підозрілих місцях замість аналізу всієї кодової бази, що мінімізує хибно-позитивні результати; по-друге, оптимальне використання AI-токенів, оскільки модель отримує лише релевантні фрагменти, а не весь репозиторій, що робить аналіз економічно ефективним навіть для великих проєктів

Після виявлення та веріфікації проблем формується загальний Production Compliance Score на основі штрафної моделі з рівнями критичності Critical, High, Medium та Low, натхненної методологією CVSS. Для кожної підтвердженої проблеми користувач може запросити автоматично згенероване виправлення, яке перед наданням проходить повторне сканування – таким чином система гарантує що запропоноване рішення не містить нових проблем.

Розроблений інструмент органічно вписується в сучасний AI-орієнтований процес розробки програмного забезпечення. В міру того як генерація коду за допомогою LLM стає стандартною практикою, автоматизована перевірка production compliance стає невід'ємною частиною циклу розробки – забезпечуючи якість та безпеку застосунків незалежно від рівня технічної підготовки їх авторів.

Список використаних джерел:

1. GitHub. Octoverse 2025: The state of open source and AI on GitHub. URL: https://github.blog/news-insights/octoverse/octoverse-2025/ (дата звернення: 11.03.2026).

2. CodeRabbit. State of AI vs Human Code Generation Report. 2025. URL: https://www.coderabbit.ai/blog/state-of-ai-vs-human-code-generation-report (дата звернення: 11.03.2026).

3. Brito T., Ferreira M., Monteiro M., Lopes P., Barros M., Santos J. F., Santos N. Study of JavaScript Static Analysis Tools for Vulnerability Detection in Node.js Packages. arXiv:2301.05097 [cs.CR]. 2023. URL: https://arxiv.org/abs/2301.05097 (дата звернення: 11.03.2026).

4. OWASP Top Ten. URL: https://owasp.org/www-project-top-ten (дата звернення: 11.03.2026).

5.FIRST.org. Common Vulnerability Scoring System v3.1: Specification Document. URL: https://www.first.org/cvss/specification-document (дата звернення: 11.03.2026).